PATENT ABSTRACTS OF JAPAN

(11)Publication number : 2002-055848
(43)Date of publication of application : 20.02.2002

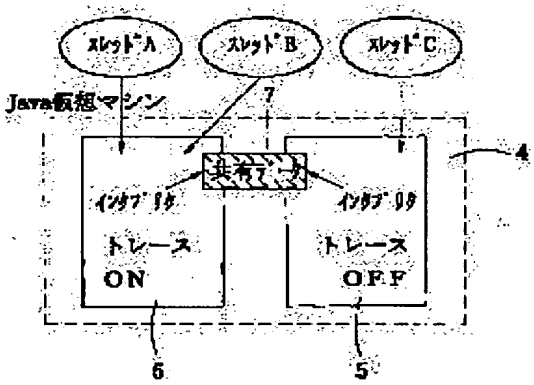
(51)Int.Cl. G06F 11/28

(21)Application number : 2000-245341
(22)Date of filing : 11.08.2000

(71)Applicant : OMRON CORP
(72)Inventor : SAIKI HIDEAKI
HIRONO MITSUAKI
KONAKA YOSHIHARU
KADOWAKI MASANORI
MURAI KENICHI

(54) PROGRAM EXECUTION PROCEDURE AND STORAGE MEDIUM WITH THE PROGRAM EXECUTION PROCEDURE STORED THEREIN

(57)Abstract:
PROBLEM TO BE SOLVED: To provide a program execution procedure which enables easy and efficient debug of an application program in a program execution environment.
SOLUTION: The program execution environment constituted of a Java virtual machine 4 is provided with a normal interpreter 5 for interpreting a thread by a normal interpreter format and executing the thread (1) and a tracing interpreter 6 for tracing the thread by executing a program having a tracing function for each line or for each instruction and outputting the contents of a memory to the external (2). Both the interpreters 5, 6 can be mutually switched, the normal interpreter 5 can execute the thread and the tracing interpreter 6 can trace and debug the thread.



LEGAL STATUS

[Date of request for examination]

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number]

[Date of registration]

[Number of appeal against examiner's decision of rejection]

[Date of requesting appeal against examiner's decision of rejection]
[Date of extinction of right]

(19)日本国特許庁（ＪＰ）　(12)公開特許公報（Ａ）　(11)特許出願公開番号
特開2002－55848
（P2002－55848A）
(43)公開日　平成14年２月20日(2002.2.20)

(51)Int.Cl.'	識別記号	ＦＩ	ﾃｰﾏｺｰﾄﾞ(参考)
Ｇ０６Ｆ　11/28	３１０ ３４０	Ｇ０６Ｆ　11/28	３１０Ａ　５Ｂ０４２ ３４０Ｃ

審査請求　未請求　請求項の数２　ＯＬ　（全 11 頁）

(21)出願番号　特願2000－245341(P2000－245341)

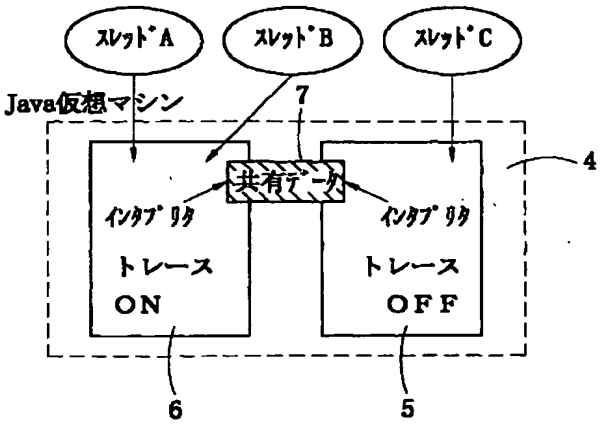
(22)出願日　平成12年８月11日(2000.8.11)

(71)出願人　000002945
オムロン株式会社
京都市下京区塩小路通堀川東入南不動堂町801番地
(72)発明者　齋木　秀明
京都府京都市下京区西洞院木津屋橋通東入ル　オムロンソフトウエア株式会社内
(72)発明者　廣野　光明
京都府京都市右京区花園土堂町10番地　オムロン株式会社内
(74)代理人　100094019
弁理士　中野　雅房

最終頁に続く

(54)【発明の名称】　プログラム実行方式及び当該プログラム実行方式を格納した記憶媒体

(57)【要約】
【課題】　プログラム実行環境において、アプリケーションプログラムを容易かつ作業効率よくデバッグできるプログラム実行方式を提供する。
【解決手段】　Ｊａｖａ仮想マシン４によって構成されたプログラム実行環境は、(1)通常のインタプリタ形式でスレッドを解釈して実行する通常インタプリタ５と、(2)トレース機能を有していてプログラムを１ラインや１命令ずつ実行し、トレースを行ってメモリの内容を外部に出力するトレース用インタプリタ６とを備えている。両インタプリタ５、６は切り替え可能となっており、通常インタプリタ５によりスレッドを実行でき、またトレース用インタプリタ６によりスレッドをトレースしてデバッグを行うことができる。



1

【特許請求の範囲】

【請求項１】　トレース機能を有する環境でプログラムを実行することができるプログラム実行環境と、トレース機能を有しない環境で実行することができるプログラム実行環境とを備え、

前記両プログラム実行環境が選択的に実行可能となっていることを特徴とするプログラム実行方式。

【請求項２】　トレース機能を有する環境でプログラムを実行することができるプログラム実行環境と、トレース機能を有しない環境で実行することができるプログラム実行環境とを備え、両プログラム実行環境が選択的に実行可能となっているプログラム実行方式を格納した記憶媒体。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】本発明は、プログラム実行方式及び当該プログラム実行方式を格納した記憶媒体に関する。具体的には、本発明は、インタプリタやオペレーティングシステムなどのプログラム実行環境、例えばＪａｖａプログラム実行環境において、アプリケーション単位あるいはスレッド単位で「トレースあり」と「トレースなし」との切換えによるトレース実行を行えるようにしたプログラム実行方式及び当該プログラム実行方式を格納した記憶媒体に関する。

【０００２】

【背景技術】従来のインタプリタやオペレーティングシステムなどのプログラム実行環境において、システム開発時もしくはシステム運用時に、アプリケーションプログラムのバグを修正し、正常に動作させるためアプリケーションプログラムをデバッグ（debug）する手段として、ソースプログラムの中にデバッグプリントを挿入することが多い。

【０００３】この方法でバグを含んだソースプログラムのデバッグ（バグ修正）を行うためには、図１に示すように、正式用のソースプログラム２にデバッグプリントを挿入してデバッグ用のソースプログラム２ａに書き直し、デバッグ用のソースプログラム２ａをコンパイルして実行ファイル３ａを作成し、プログラム実行環境１でデバッグ用の実行ファイル３ａを実行することによってバグ取りを行う。そして、デバッグ用のソースプログラム２ａのバグ修正が完了したら、正式用のソースプログラム２も同様に修正し、実行ファイル３にコンパイルする。

【０００４】しかし、デバッグプリントを挿入する方法では、図１に示すように、正式用とデバッグ用の最低２種類のソースプログラム２、２ａが混在することになるので、同一のソースプログラムについて２種類以上のプログラムを記述しなければならず、バグ修正作業が非常に乱雑となり、コード解析にも支障をきたすことがある。

2

【０００５】また、ソースプログラムにデバッグプリントを挿入することによってデバッグする方法では、デバッグ出力が待ち状態になる可能性があるため、デバッグプリントを入れることで処理のタイミングが変わる可能性がある。例えば、図２（ａ）に示すように、正式用のソースプログラム２が、スレッド（Thread；プログラム実行管理単位）ＡとスレッドＢからなり、スレッドＡが処理ＳＡ１、ＳＡ２、ＳＡ３によって、スレッドＢが処理ＳＢ１、ＳＢ２、ＳＢ３によって構成されており、スレッドＡの処理ＳＡ１、ＳＡ２、ＳＡ３が順次実行された後、スレッドＢの処理に制御が移り、処理ＳＢ１、ＳＢ２、ＳＢ３が順次実行されるとする。このとき、図２（ｂ）に示すように、デバッグ用のソースプログラム２ａにおいて、スレッドＡの処理ＳＡ２と処理ＳＡ３との間にデバッグプリントが挿入されたとすると、スレッドＡにおける処理ＳＡ１、ＳＡ２に続いてデバッグプリントの処理が実行される。ここで、デバッグ出力が待ち状態になってしまったりすると、制御がスレッドＢへ移り、スレッドＢで処理ＳＢ１、ＳＢ２、ＳＢ３が順次実行された後、再び制御がスレッドＡに戻って処理ＳＡ３が実行され、正式用のソースプログラム２と実行順序が変わってしまうような不都合が生じる恐れがある。

【０００６】さらに、デバッグプリントを挿入することによってデバッグする方法では、ある現象を把握するためのタイミングのコツを知る必要があり、開発者にとって効率的に作業が行えないという問題点があり、デバッグに要する時間が非常に長いものとなり、非効率的であった。

【０００７】また、デバッグプリントを挿入する以外のデバッグ手段としては、デバッガを利用する方法もある。しかし、デバッガを用いる方法では、デバッガを使用するための知識が必要となり、また組み込み機器のためのソフトウエアであれば、ハードウェア毎に設定が異なり、対象とするハードウェアに関する知識が必要となるため、デバッガを習熟するための時間やデバッグに至るまでの時間が長く掛かる不都合がある。

【０００８】このデバッガの問題を解決するための公知技術としては、インタプリタ言語であるＢＡＳＩＣにトレース機能が存在し、この仕組みを取り入れることが考えられる。しかし、ＢＡＳＩＣのトレース機能は、ソースプログラムにトレース開始／終了のステートメントを挿入することで実現できるため、図３（ａ）（ｂ）に示すように、デバッグ用のソースプログラム２ａでは、正式用のソースプログラム２に対してトレースの開始位置にトレースＯＮを、トレースの終了位置にトレースＯＦＦをそれぞれ挿入する必要がある。よって、結局ソースプログラムを書き直す必要があり、デバッグプリントを挿入するのと同様、作業時間と手間が掛かってしまうことになる。

【０００９】

【発明の開示】本発明の目的とするところは、プログラム実行環境において、アプリケーションプログラムを容易かつ作業効率よくデバッグできるプログラム実行方式を提供することにある。また、そのプログラム実行方式を格納した記憶媒体を提供することにある。

【００１０】本発明にかかるプログラム実行方式は、トレース機能を有する環境でプログラムを実行することができるプログラム実行環境と、トレース機能を有しない環境で実行することができるプログラム実行環境とを備え、前記両プログラム実行環境が選択的に実行可能となっていることを特徴としている。

【００１１】また、本発明にかかる記憶媒体は、トレース機能を有する環境でプログラムを実行することができるプログラム実行環境と、トレース機能を有しない環境で実行することができるプログラム実行環境とを備え、両プログラム実行環境が選択的に実行可能となっているプログラム実行方式を格納したものである。ここで記憶媒体としては、特にその種類は限定されることはなく、ハードディスク、ＣＤ類、ＤＶＤ、ＭＯ、各種ＩＣメモリなどが該当する。

【００１２】本発明にあっては、複数のプログラム実行環境が切り換え可能となっており、トレース機能を有するプログラム実行環境とトレース機能を有しないプログラム実行環境のいずれかでプログラムを実行可能となっているため、プログラムのソースファイルにデバッグプリント等のトレースのための仕組みが必要なくなる。従って、プログラムのデバッグを行うため、そのソースプログラムを書き直したり、デバッグプリントを挿入したりする必要が無くなるので、あるいはソースプログラムの書き替え作業をわずかなものにすることができるので、プログラムを容易かつ作業効率よくデバッグすることができる。よって、インタプリタやアプリケーションプログラムなどのプログラム実行環境において、システム開発時やシステム運用時に、デバッグのための作業時間を削減でき、また、ソースプログラムを書き直す必要がないので、デバッグ時のコード解析も簡単になる。

【００１３】また、本発明の方式によれば、アプリケーションプログラムの実行中でもトレース環境が存在するため、トレース実行の切換え自由度が高くなる。例えば、アプリケーションプログラム起動前における切換え、アプリケーションプログラム実行中での切換え、アプリケーションプログラム内のスレッド毎の切換え等が可能になる。

【００１４】

【発明の実施の形態】（第１の実施形態）プログラム実行環境としてＪａｖａプログラム実行機構（Ｊａｖａ仮想マシン）の場合を例として、以下本発明の一実施形態を説明する。従来のＪａｖａ仮想マシンでは、単一のプログラム実行環境によってスレッドを実行しているが、本発明にかかるＪａｖａ仮想マシンは、複数のプログラミング実行環境を有している。

【００１５】図４はＪａｖａ仮想マシン４を用いた本発明にかかるＪａｖａプログラム実行環境の概要を表している。この実施形態では、１つのＪａｖａ仮想マシン４（Virtual Machine；プログラムから生成されたバイトコード、すなわち中間ファイルを実行するための環境）に対し、バイトコードを１命令ずつ逐次解釈しながら実行するためのインタプリタ５、６を２つ備え、両インタプリタ５、６で共有するデータ領域７を有している。一方のインタプリタ５（以下、通常インタプリタという。）はトレース機能を備えておらず、通常のインタプリタ形式でスレッドを解釈して実行する。他方のインタプリタ６（以下、トレース用インタプリタという。）はトレース機能を備えており、プログラムを１ラインや１命令ずつ実行し、トレースを行ってメモリの内容を外部に出力することができる。トレース結果の出力は、ＲＳ－２３２Ｃのようなパラレル出力やシリアル出力から出力させることができる。トレースの出力単位は、バイトコード単位、メソッド（関数）呼び出し単位のどちらでも出力可能である。

【００１６】このＪａｖａ仮想マシン４においては、両インタプリタ５、６を選択的に切り換えてスレッドを実行することができる。インタプリタ５、６の切換え方法ないし切換え時期としては種々用意しておくことができる。ここでは、例として、(1)アプリケーションプログラムが起動する前におけるインタプリタの切換え機能、(2)アプリケーションプログラム実行時におけるインタプリタの切換え機能、(3)スレッド毎のインタプリタ切換え機能、(4)局所的なトレースの切換え機能について説明する。Ｊａｖａ仮想マシン４は、これらのインタプリタ又はトレースの切換え機能のうち１つだけを備えていてもよく、複数備えていてもよい。

【００１７】まず、アプリケーションプログラム起動前のインタプリタ切換えについて説明する。このＪａｖａ仮想マシン４には、スレッドを管理するための構造体が存在しており、そのメンバーとしてインタプリタ切換えのためのフラグが用意されている。そして、アプリケーションプログラム起動時には、当該フラグによって、２つのインタプリタのうち通常インタプリタ５で実行するか、トレース用インタプリタ６で実行するかが決まる。一方、Ｊａｖａ仮想マシン４またはアプリケーションプログラムには、トレース有り、トレース無しのいずれかを設定するための環境変数（インタプリタを実行するための変数）を設けてあり、コマンド入力等によって環境変数をトレース有り、トレース無しのいずれかに設定できるようになっている。

【００１８】しかして、このプログラム実行環境において、コマンド入力等によって環境変数をトレース有りに設定してアプリケーションプログラムを起動すると、図５のフロー図に示すように、環境変数によって上記フラ

グがトレース有りに設定され（ステップＳ１でトレース設定オン）、アプリケーションプログラムで使用されるスレッド全てに対応づけられる。従って、プログラム実行環境がトレース用インタプリタ６に設定され（ステップＳ２）、いずれのスレッドもトレース用インタプリタ６で実行され、トレース結果が外部へ出力される（ステップＳ３）。

【００１９】また、コマンド入力等によって環境変数をトレース無しに設定してアプリケーションプログラムを起動すると、環境変数によって上記フラグがトレース無しに設定され（図５のステップＳ１でトレース設定ＯＦＦ）、アプリケーションプログラムで使用されるスレッド全てに対応づけられる。よって、プログラム実行環境が通常インタプリタ５に設定され（ステップＳ４）、いずれのスレッドもトレース機能無しの通常インタプリタ５で実行される（ステップＳ５）。

【００２０】このようにして２つのインタプリタ５、６を切換えるようにすれば、Ｊａｖａ仮想マシン４が動作する実行環境でトレースの有無を設定できるので、デバッグを行なうためにソースプログラムを変更する必要がなく、アプリケーションプログラムを容易かつ作業効率よくデバッグできる。

【００２１】次に、アプリケーションプログラム実行時におけるインタプリタ切換え機能について説明する。アプリケーションプログラムの実行時（起動後）にインタプリタを切換える機能を持たせる場合には、トレース有り、トレース無しのいずれかを設定するための環境変数（インタプリタを実行するための変数）を設け、インタプリタ実行中に、コマンド入力等によって環境変数をトレース有り、トレース無しのいずれかに再設定できるようにする。この環境変数は、初期値ではトレース無しに設定されている。

【００２２】しかして、アプリケーションプログラムが起動されると、図６のフロー図に示すように、環境変数がトレース無しに設定されているので、通常インタプリタ５でスレッドを実行可能な状態で起動する（ステップＳ１１）。このときには、まだコマンド入力はないので（ステップＳ１３でオフの場合）、トレース機能がオフの通常インタプリタ５が作動し（ステップｓ１４）、トレース無しでスレッドが実行される（ステップＳ１５）。

【００２３】アプリケーションプログラムが終了していなければ（ステップＳ１８でＮＯの場合）、ステップＳ１３～Ｓ１８の処理が繰り返されるが、実行途中でコマンド入力（ステップＳ１２）によってトレース機能がオンとなるように環境変数が書き替えられた場合（ステップＳ１３でオンの場合）には、プログラム実行環境がトレース用インタープリタに切り替えられ（ステップＳ１６）、トレース用インタプリタ６で各スレッドを実行しながらトレースが実行され、トレース結果が外部へ出力

される（ステップＳ１７）。

【００２４】また、トレース機能がオンになった状態でスレッドが実行されているときに、コマンド入力（ステップＳ１２）によってトレース機能がオフに切り替えられた場合（ステップＳ１３でオフの場合）には、再びプログラム実行環境が通常インタープリタに切り替えられ（ステップＳ１４）、トレース機能のない通常インタプリタ５でスレッドが実行される（ステップＳ１５）。

【００２５】このようにしてアプリケーションプログラムの実行中に２つのインタプリタを切換えられるようにしても、Ｊａｖａが動作する実行環境でトレースの有無を設定できるので、デバッグを行なうためにソースプログラムを変更する必要がなく、アプリケーションプログラムを容易かつ作業効率よくデバッグできる。

【００２６】次に、スレッド毎のインタプリタ切換え機能について説明する。アプリケーションプログラムに存在するスレッド個々に対して、それぞれトレースを実行するか否かを設定できるようにする場合には、トレース用インタプリタ６をオンにする機能を組み込まれたトレース専用のスレッドクラス８を提供しておく。そして、図７のスレッドＣのように、トレースを行わないスレッドに対しては、当該トレース用スレッドクラス８（または、当該トレース用クラス８を継承しているスレッドクラス）を継承させることなく通常のスレッドクラスを継承してスレッドを生成させる。一方、図７のスレッドＡ、Ｂのように、トレースを行いたいスレッドに対しては、トレース用スレッドクラス８（または、トレース用スレッドクラス８を継承したスレッドクラス）を継承させておけば、実行時にトレース用インタプリタ６が実行されるよう、スレッド生成時に内部でスレッド切換えの設定が行われる。しかして、トレース用スレッドクラス８を承継していないスレッドＣが実行される場合には、Ｊａｖａ仮想マシン４が通常インタプリタ５に設定され、通常インタプリタ５によって当該スレッドが実行され、トレースは行われない。これに対し、トレース用スレッドクラス８を承継しているスレッドＡ、Ｂが実行された場合には、当該スレッドＡ、Ｂはトレース用インタプリタ６によって実行され、トレース結果が出力される。

【００２７】スレッド毎のトレース切換えを行うことができるようにした場合の具体的なスレッド実行手順は、図８に示すようになる。すなわち、アプリケーションプログラム起動前に環境変数がトレース無しに設定されている場合には、トレース設定の有無を判別することによって（ステップＳ２１）トレース設定ＯＦＦと判断され、トレース用スレッドクラス８を継承しているか否かにかかわりなくＪａｖａ仮想マシン４は通常インタプリタ５に設定され（ステップＳ２３）、トレース無しでスレッドが実行される（ステップＳ２４）。

【００２８】また、アプリケーションプログラム起動前

に環境変数がトレース有りに設定されている場合には、トレース設定の有無を判別することによって（ステップＳ２１）トレース設定ＯＮと判断された後、さらに、実行しようとするスレッドがトレース用スレッドクラス８を継承しているか否か判別される（ステップＳ２２）。このとき当該スレッドがトレース用スレッドクラス８を継承していない場合には、Ｊａｖａ仮想マシン４は通常インタプリタ５に設定され（ステップＳ２３）、トレースなしでスレッドが実行される（ステップＳ２４）。これに対し、スレッドがトレース用スレッドクラス８を継承している場合には、Ｊａｖａ仮想マシン４はトレース用インタプリタ６に設定され（ステップＳ２５）、当該スレッドはトレース用インタプリタ６で解釈実行され、トレース出力が外部へ出力される（ステップＳ２６）。

【００２９】このような方法では、スレッドの継承元をトレース用のスレッドクラスと、トレース用インタプリタ６をオンにする機能のない通常のスレッドクラスとに変更するだけでよく、ソースプログラム自体の変更作業を最小の手間にする必要がないので、アプリケーションプログラムを容易かつ作業効率よくデバッグできる。あるいは、スレッドの継承元のスレッドクラスにトレース有りとトレース無しを切り替えるためのプロパティ等を組み込んでおき、コマンドによって切り替えられるようにしてもよい。

【００３０】次に、局所的なトレースの切換え機能、すなわちアプリケーションプログラムやスレッドの一部分だけのトレース切換え機能について説明する。この場合には、トレース専用のスレッドクラスを提供しておき、そのスレッドクラスもしくは当該スレッドクラスを継承したスレッドクラスでは、トレース用インタプリタ６をトレース実行モードとトレース中止モードとに切り替えるためのメソッド（関数）を利用できるようにする。

【００３１】図９は局所的なトレースの切り替えについて説明する図であって、スレッドＡ、Ｂ、Ｃのソースプログラム９ａ、９ｂ、９ｃに含まれている”トレースＯＮ”は、トレース用インタプリタ６をトレース実行モードに切り替えるためのメソッド、”トレースＯＦＦ”は、トレース用インタプリタ６をトレース中止モードに切り替えるためのメソッドであって、いずれもメソッド呼び出し時に内部で環境変数の設定を行うことにより、トレース用スレッド６をトレースを実行する状態とトレースを実行しない状態とに設定するものである。また、トレース有りのインタプリタ６は、初期状態がトレース中止モードで実行されるとする。

【００３２】しかして、これらのソースプログラム９ａ、９ｂ、９ｃをコンパイルした実行ファイルが、実行される場合には、図１０に示すフロー図に従う。すなわち、図９のスレッドＡ、Ｂの場合のように、コマンド入力等によって環境変数をトレース有りに設定してアプリケーションプログラムを起動すると（ステップＳ３１でトレース設定オン）、環境変数の設定によってトレース用インタプリタ６が実行される（ステップＳ３２）。トレース用インタプリタ６が実行されると、トレース用フラグは初期値０に設定され（ステップＳ３３）、トレースメソッド（”トレースＯＮ”、”トレースＯＦＦ”）か通常の処理コマンドから判別する（ステップＳ３４）。このトレース用フラグは、トレース用インタプリタ６をトレース実行モードとトレース中止モードに設定するものであって、トレース用フラグが０の場合にはトレース中止モードに設定され、１の場合にはトレース実行モードに設定される。

【００３３】トレース用フラグの初期値は０となっているので、スレッドＡ、Ｂの処理ＳＡ１、ＳＢ１は、トレース中止モードでスレッドが実行されるが（ステップＳ３４、Ｓ３８、Ｓ４０）、メソッドが見つかると（ステップＳ３４）、そのメソッドが”トレースＯＮ”か”トレースＯＦＦ”か判別し（ステップＳ３５）、”トレースＯＮ”であればトレースフラグを１に設定して次の処理に進み（ステップＳ３６）、”トレースＯＦＦ”であれば、トレースフラグを０に設定して次の処理に進む（ステップＳ３７）。

【００３４】このとき”トレースＯＮ”のメソッドであれば、トレースフラグは１であると判別される（ステップＳ３８）ので、次の処理ＳＡ２、ＳＡ３；ＳＢ２、ＳＢ３はトレース実行モードとなり、トレース用インタプリタ６によりスレッドを実行されながらトレースが実行される（ステップＳ３９）。

【００３５】ついで、”トレースＯＦＦ”のメソッドが実行されると、トレースフラグが０に設定される（ステップＳ３４、Ｓ３５、Ｓ３７）ので、１であると判別される（ステップＳ３８）ので、トレース中止モードに戻り、次の処理ＳＡ４、ＳＢ４はトレース用インタプリタ６によりスレッドのみ実行される（ステップＳ４０）。そして、すべての処理が実行されると、スレッドＡ、Ｂの実行が終了する（ステップＳ４１）。

【００３６】次に、図９のスレッドＣの場合のように、コマンド入力等によって環境変数をトレース無しに設定してアプリケーションプログラムを起動すると（ステップＳ３１でトレース設定オフ）、環境変数の設定によって通常インタプリタ５が起動される（ステップＳ４２）。そして、スレッドＣがトレース用のメソッドを含んでいるか否かにかかわらず、全ての処理ＳＣ１～ＳＣ４が通常インタプリタ５により実行される（ステップＳ４３、Ｓ４４）。

【００３７】この場合には、スレッドにトレース用のメソッドを含んでいても通常インタプリタを実行することにより通常のスレッドのように実行することができるので、正式用のソースプログラムに当初からトレース用のメソッドを挿入しておけば、正式用のソースプログラムとデバッグ用のソースプログラムを別個に作成する必要

9

が無くなり、アプリケーションプログラムを容易かつ作業効率よくデバッグできる。

【００３８】本発明にあっては、上記のようにトレース有りのプログラム実行環境とトレース無しのプログラム実行環境とを備えているので、図１１に示すように、ソースプログラム２をコンパイルして実行ファイル３を作成した後、トレース無しのプログラム実行環境で実行ファイル３（スレッド）を実行させることができ、またトレース有りのプログラム実行環境で実行ファイル３をデバッグすることができる。よって、従来例のようにデバッグプリントを挿入されたソースプログラム２ａを別途作成してデバッグ用の実行ファイル３ａを作成する必要が無くなる。このため、本発明によれば、デバッグのための作業時間を大幅に短縮することができる。例えば、デバッグプリント挿入によってデバッグを行う方法では、図１２に示すように、正式用のソースファイルにデバッグプリントを挿入するのに要する時間が１ファイルあたり１８０秒、１ファイルのコンパイル時間が１０秒とし、全部で２００ファイルとすると、デバッグ用の実行ファイルを作成するのに、約６００分（約１０時間）掛かることになる。これに対し、アプリケーションプログラム実行前にプログラム実行環境のトレース有り、無しを切り替える本発明方法では、正式用の実行ファイルをそのまま使用できるので、デバッグのための実行ファイルを作成する時間は必要ない。よって、本発明の方法によれば、デバッグ作業に要する手間と時間を大幅に軽減することができる。

【００３９】

【発明の効果】本発明によれば、トレース機能を有するプログラム実行環境とトレース機能を有しないプログラム実行環境のいずれかでプログラムを実行可能となっているため、プログラムのソースファイルにデバッグプリント等のトレースのための仕組みが必要いので、プログラムのデバッグを行うためにソースプログラムを書き直したり、デバッグプリントを挿入したりする必要が無くなり、プログラムを容易かつ作業効率よくデバッグすることができるようになる。

10

【図面の簡単な説明】

【図１】従来のデバッグ方法の一例を示す概略図である。

【図２】（ａ）は同上のデバッグ方法に用いられている正式用のソースプログラムを示す図、（ｂ）はデバッグ用のソースプログラムを示す図である。

【図３】トレースによりデバッグを行う従来方法を説明する図であって、（ａ）は正式用のソースプログラムを示す図、（ｂ）はデバッグ用のソースプログラムを示す図である。

【図４】本発明の一実施形態にかかるＪａｖａ仮想マシンの構成を示す概略図である。

【図５】同上のＪａｖａ仮想マシンにおいて、アプリケーションプログラム起動前にインタプリタを切り替える手順を示すフロー図である。

【図６】図４のＪａｖａ仮想マシンにおいて、アプリケーションプログラム実行中にインタプリタを切換える手順を示すフロー図である。

【図７】図４のＪａｖａ仮想マシンにおいて、スレッド毎にトレースの有無を切換える方法を説明する図である。

【図８】同上のＪａｖａ仮想マシンにおいて、スレッド毎にインタプリタを切り替える手順を示すフロー図である。

【図９】図４のＪａｖａ仮想マシンにおいて、局所的にトレースの有無を切り替える方法を説明する図である。

【図１０】同上のＪａｖａ仮想マシンにおいて、局所的なインタプリタの切換え手順を示すフロー図である。

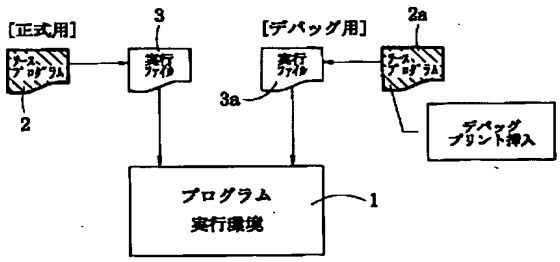
【図１１】本発明のデバッグ方法と従来例のデバッグ方法を比較して説明する図である。

【図１２】本発明のデバッグ方法に要する時間と従来例のデバッグ方法に要する時間を比較して示す図である。

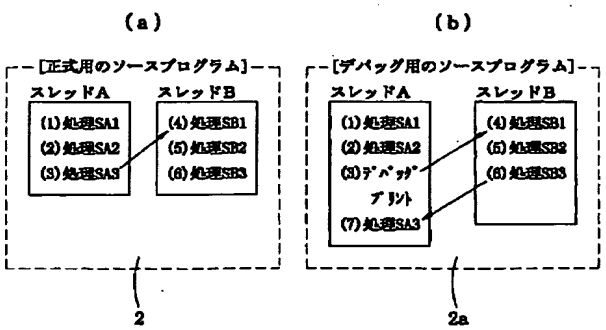
【符号の説明】

４　Ｊａｖａ仮想マシン
５　通常インタプリタ
６　トレース用インタプリタ

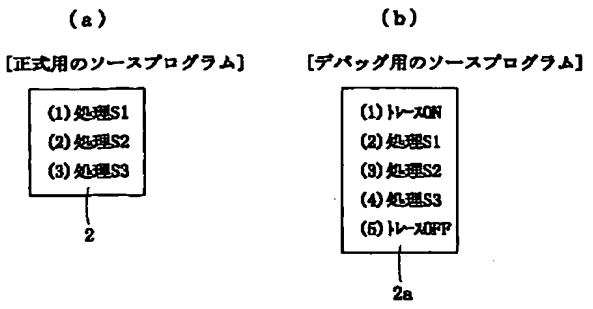
【図１】



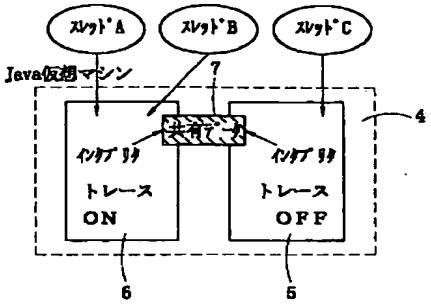
【図２】



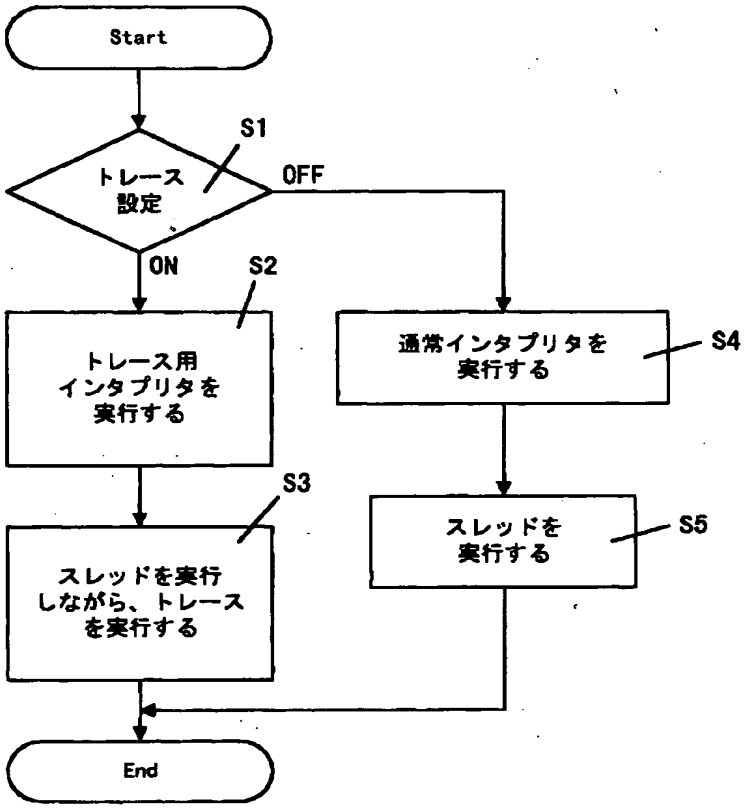
【図３】



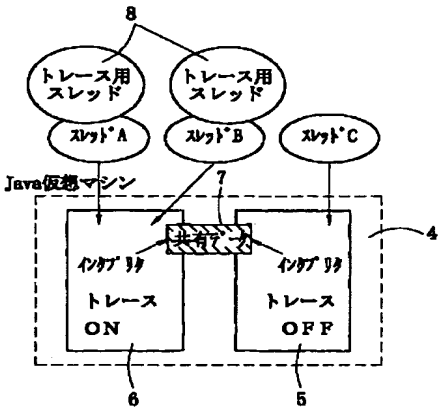
【図４】



【図５】



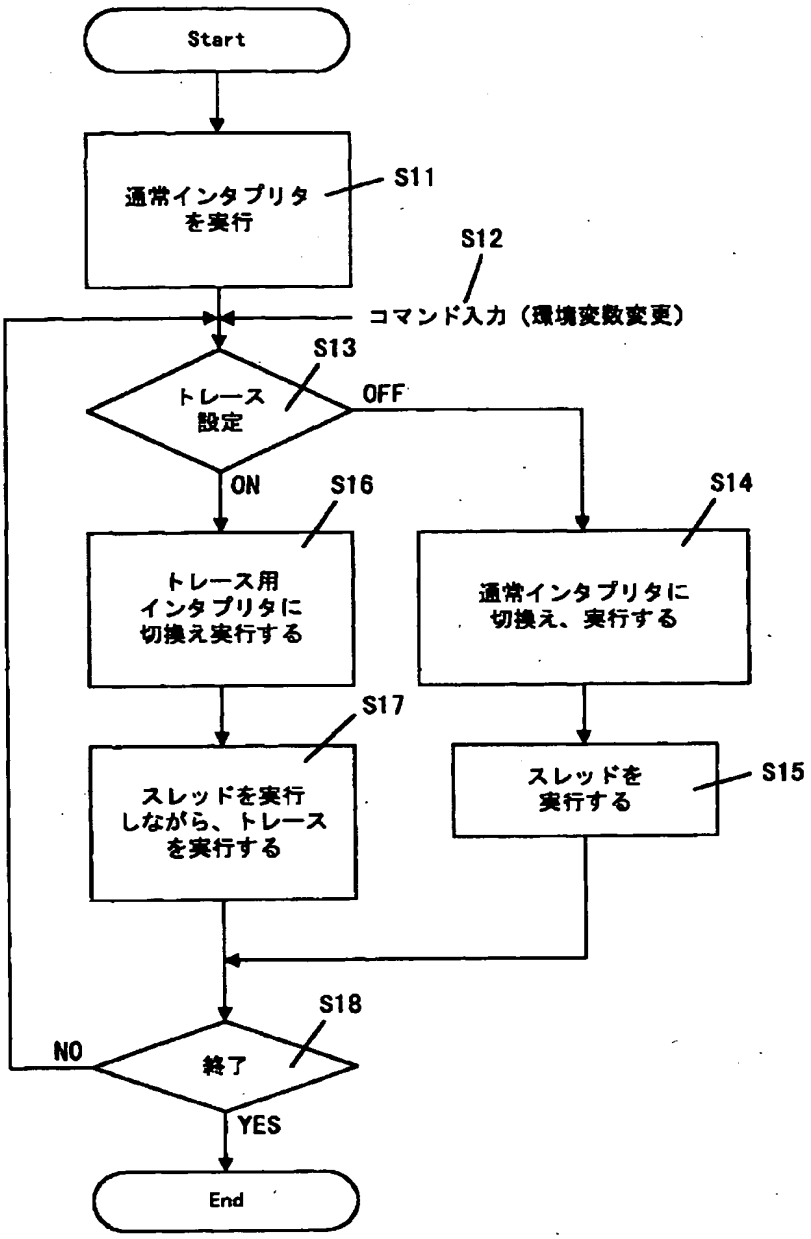
【図７】



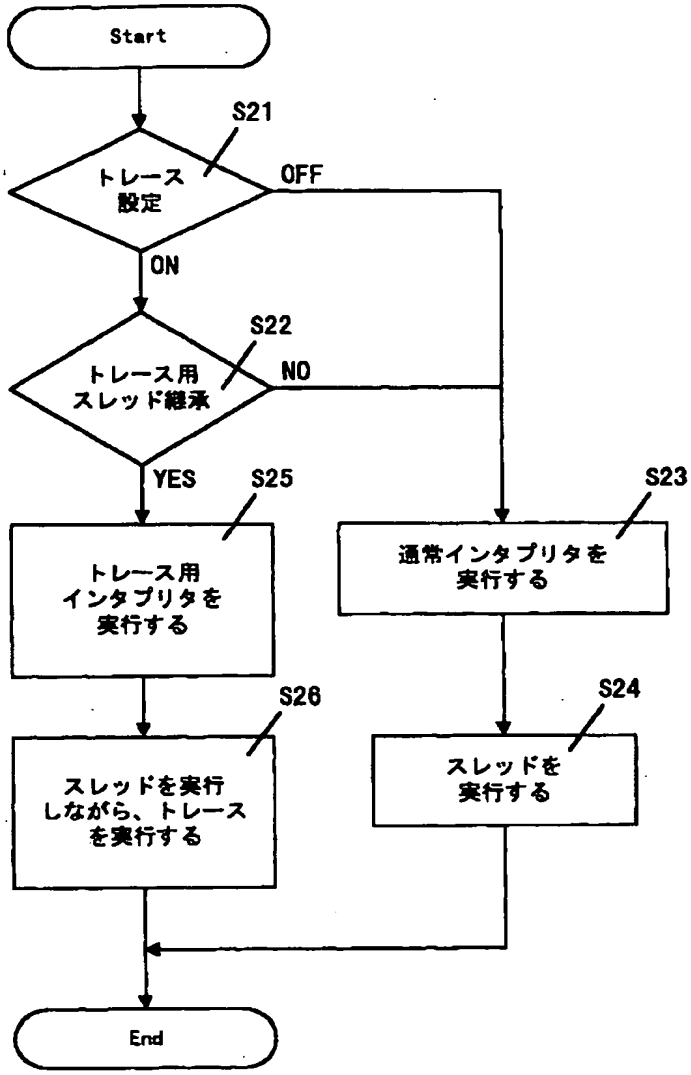
【図１２】

	１ﾌｧｲﾙにあたるﾃﾞﾊﾞｯｸﾞﾌﾟﾘﾝﾄ挿入時間	コンパイル時間	ﾌｧｲﾙ数	総時間
従来	180秒	10秒	200	約600分
本発明	0秒	0秒	200	0分

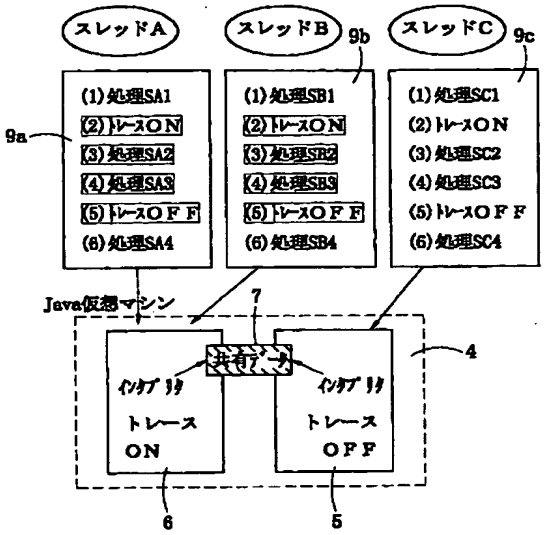
【図６】


Start
通常インタプリタを実行
S11
S12
コマンド入力（環境変数変更）
S13
トレース設定
OFF
ON
S16
S14
トレース用インタプリタに切換え実行する
通常インタプリタに切換え、実行する
S17
スレッドを実行しながら、トレースを実行する
スレッドを実行する
S15
S18
NO
終了
YES
End

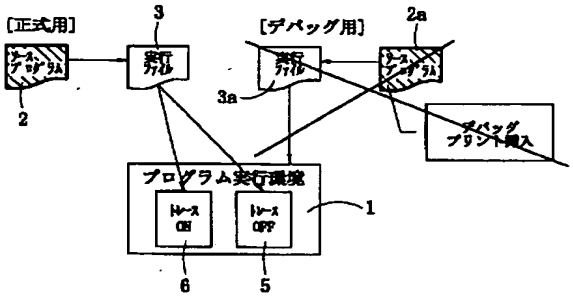
【図８】


Start
S21
トレース
設定
OFF
ON
S22
トレース用
スレッド継承
NO
YES
S25
トレース用
インタプリタを
実行する
S23
通常インタプリタを
実行する
S26
スレッドを実行
しながら、トレース
を実行する
S24
スレッドを
実行する
End

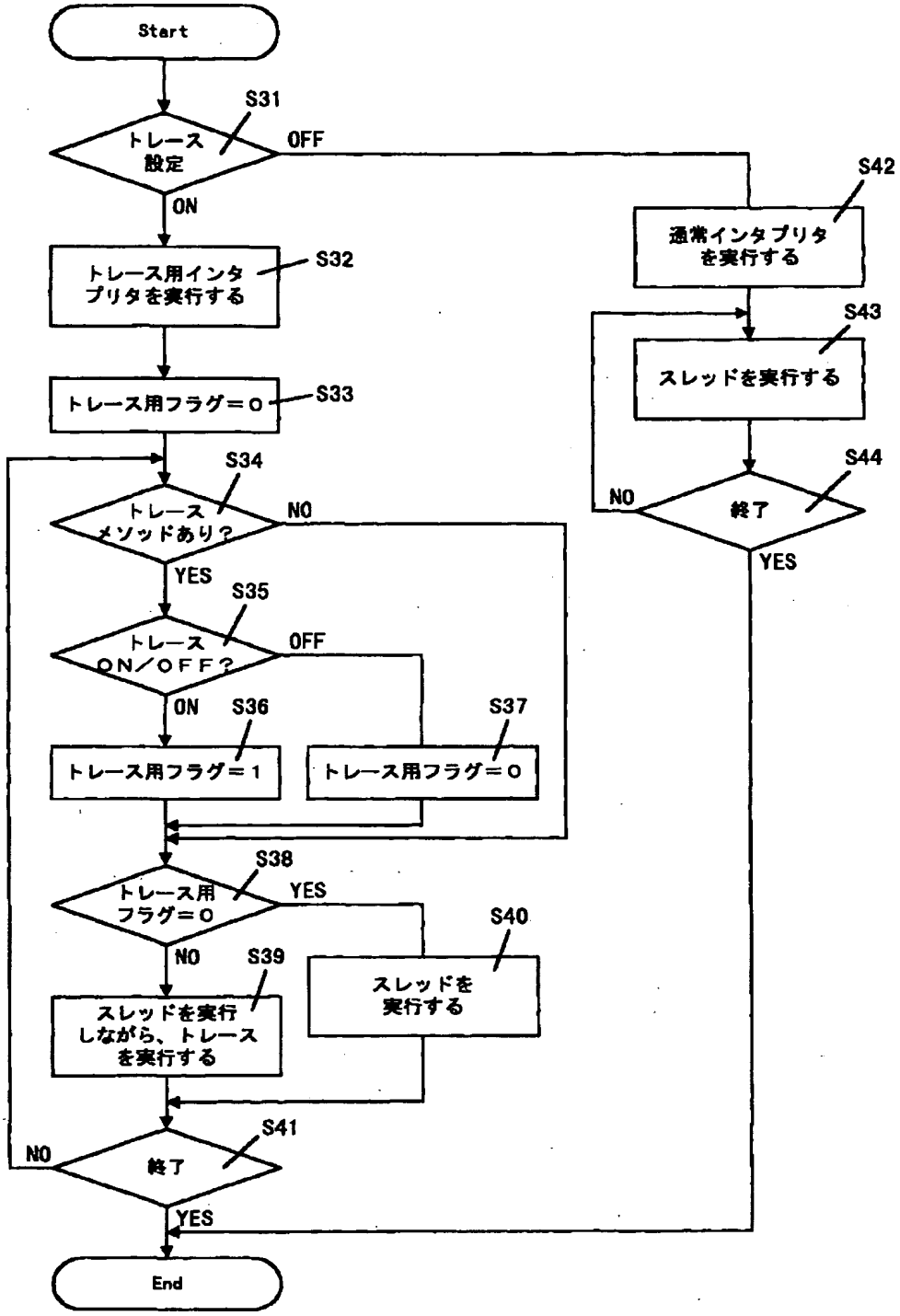
【図９】


スレッドA
スレッドB
9b
スレッドC
9c
9a
(1)処理SA1
(2)トレースON
(3)処理SA2
(4)処理SA3
(5)トレースOFF
(6)処理SA4
(1)処理SB1
(2)トレースON
(3)処理SB2
(4)処理SB3
(5)トレースOFF
(6)処理SB4
(1)処理SC1
(2)トレースON
(3)処理SC2
(4)処理SC3
(5)トレースOFF
(6)処理SC4
Java仮想マシン
7
共有データ
インタプリタ
インタプリタ
トレース
ON
トレース
OFF
4
6
5

【図１１】


[正式用]
3
[デバッグ用]
2a
2
3a
デバッグ
プリント挿入
プログラム実行環境
トレース
ON
トレース
OFF
1
6
5

【図１０】


Start
S31
トレース設定
OFF
ON
S32
トレース用インタプリタを実行する
S33
トレース用フラグ＝０
S34
トレースメソッドあり？
NO
YES
S35
トレースON／OFF？
OFF
ON
S36
トレース用フラグ＝１
S37
トレース用フラグ＝０
S38
トレース用フラグ＝０
YES
NO
S39
スレッドを実行しながら、トレースを実行する
S40
スレッドを実行する
S41
終了
NO
YES
End
S42
通常インタプリタを実行する
S43
スレッドを実行する
S44
終了
NO
YES

フロントページの続き

(72)発明者　小中　義治
京都府京都市右京区花園土堂町10番地　オムロン株式会社内
(72)発明者　門脇　正規
京都府京都市右京区花園土堂町10番地　オムロン株式会社内
(72)発明者　村井　謙一
京都府京都市右京区花園土堂町10番地　オムロン株式会社内

Ｆターム(参考)　5B042 GA08 GA23 HH07 HH30 MA10 MA20